Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Pato Branco
Disciplina: Estruturas de Dados, Pesquisa e Ordenação
Professora: Emanoeli Madalosso
Curso: Tecnologia em Análise e Desenvolvimento de Sistemas

Avaliação 2 - 01/11/2019

1) **(2,0 pontos)** Escreva uma função "inverte_lista", que inverte os itens de uma lista ESTÁTICA (sem o auxílio de uma lista auxiliar). Exemplo:

```
Lista original: 10, 25, 13, 4, 50
Lista invertida: 50, 4, 13, 25, 10
```

2) **(3,0 pontos)** Escreva um programa para ler uma sequência (não necessariamente ordenada) de números inteiros positivos e armazená-los em um vetor *v*. O último número lido é o zero, o qual não deve fazer parte dos valores de *v*. Além desses, mais um único valor deve ser lido, representando o limite de uma soma.

Após a leitura dos dados, o programa deve imprimir apenas os últimos números de cada subsequência de *v* cuja soma de seus valores ultrapassa o limite de soma. Uma vez ultrapassado esse limite, uma nova subsequência deve ser iniciada a partir do valor que segue o último da subsequência identificada no momento. Ao final do processamento completo, seu programa deverá imprimir o valores que ultrapassam o limite da soma, ao contrário da forma que é processado.

ATENÇÃO: os valores digitados podem ser guardados em um vetor, mas os resultados NÃO podem ser guardados em um vetor, eles devem ser armazenados em uma das estruturas vistas em aula. Exemplo:

```
Valores digitados: 33 51 23 94 66 28 11 73 19 8 31 0
Limite da soma: 90
Saída:
19 28 94 23
```

Neste exemplo, o valor 23 é o último da subsequência [33, 51, 23], cuja soma é 107, ultrapassando o limite de soma 90, sendo assim deverá ser impresso por último. Com isso, logo depois do valor 23, tem início a verificação da soma dos valores de uma nova subsequência que começa com o número 94. De cara, o 94 já ultrapassa o limite 90, sendo o penúltimo número a ser impresso, e assim por diante até que todo o vetor *v* seja processado.

3) **(4,0 pontos)** Um dilúvio está a caminho, e você foi encarregado de ser o porteiro da arca de Noé versão 2.0. Toda vez que um animal entrar na arca, você deve cadastrá-lo em uma lista ENCADEADA (simples ou dupla, se preferir). Entretanto, Noé já está consciente que a arca moderna não vai aguentar todos os animais (a madeira hoje em dia é de péssima qualidade). Sendo assim, além do cadastro, você deve conseguir descobrir facilmente qual é o animal mais pesado e também o mais

leve, para que estes sejam jogados na água sempre que for necessário aliviar o peso da arca.

Para cada animal guarde seu código e seu peso. Por exemplo:

```
struct item {
	int codigo;
	float peso;
};
```

Além do ponteiro de início (e, se desejável, um de fim), sua lista deve ter sempre atualizados 2 ponteiros para, respectivamente, o animal mais pesado e o mais animal mais leve. Exemplo:

```
struct lista {
	Celula *primeira;
	Celula *mais_pesado;
	Celula *mais_leve;
};
```

Crie uma função “diminuir_ocupacao” que: imprime o código do mais pesado e do mais leve, remove eles da lista e atualiza os ponteiros “mais_pesado” e “mais_neve”. Dica: você pode chamar a função remove_item() que foi implementada em aula dentro desta função, não precisa fazer toda a lógica de remover do zero.